

USAspending Database Archive - Recommended Download and Restoration Process


Our entire database as a PostgreSQL archive — the most complete download option available for advanced users.

Overview

This document provides instructions for downloading and restoring the USAspending.gov database archive. The archive provides the complete database powering USAspending.gov, which includes data on all spending by the federal government, including contracts, grants, loans, employee salaries, and more.

Prerequisites

In order to be successful using this document, there are several things that you must have set up ahead of time.

1. **Hard Drive Space.**  The database is **large**, and will consume a lot of disk space. Ensure you have disk space to accommodate the following:
 - Over 51 gigabytes of hard drive space to store the uncompressed (unzipped) database archive file that was downloaded
 - Over 1.5 terabytes of additional hard drive space to store the fully restored database, with all materialized view data refreshed.
 - **NOTE:** The base tables represent about ½ of this total database size. The other ½ of this total database size is the materialized view data and their indexes. There are options in the steps below to skip generation of materialized views and leave base tables only. This will affect your ability to run the USAspending API, as it relies on these materialized views, but will still provide the complete set of USAspending's data across the non-materialized tables.
 - **NOTE:** These are approximate sizes at the date this document was last revised (July 2019). Sizes will continue to increase beyond what is referenced above as our database continues to grow over time.
2. **PostgreSQL.** The database archive can be restored to a PostgreSQL database using `pg_restore` which is bundled with PostgreSQL. It has been tested with PostgreSQL 10.6 .
3. **Postgres Contribution Modules.** The database uses a handful of PostgreSQL EXTENSIONS , and therefore your installation of PostgreSQL will need to have all of the modules required by those extensions. These can be added-on in a package often named something like `postgres-contrib*` as part of the package manager for your platform. See the PostgreSQL download page for where to find the contributions modules:
<https://www.postgresql.org/download/>.
 - TIP: You can see all the extensions used with `pg_restore --list usaspending-db-dump-dir | grep EXTENSION` once the archive is unzipped.
4. **root Superuser.** A user named `root` on your PostgreSQL server with `SUPERUSER` privileges. The archive was dumped from a generic database where objects were owned by a user named `root` , and it will simplify the restore if that same user exists on the server this database will be restored to. If this can't be done, then `pg_restore` should be invoked with the `--no-owner` flag. Consult the PostgreSQL [createuser documentation](#) for how to create a user or alter permissions.
5. **Time.** Given the size of data to be restored, it can take many hours to complete a full database restore. If you are restoring to a remote location, make sure you have a way to keep the restore process running in the background without your remote connection timing out.
6. **Tools.** We mention several tools we use in these instructions. This is not intended to be an endorsement of any specific tools.

Steps

1. Download the Archive File

The archive file is a `.zip` file that can be downloaded from the web. The hyperlink for downloading that file should be provided in the same place that this document was linked.

Click the link and save the file onto your hard drive. If you are placing the file on a remote machine, you may want to use tools like `wget`, `cURL`, or `Invoke-WebRequest` from a command shell to pull the file down to the filesystem.

2. Unzip the File and Inspect the Archive

Once downloaded, unzip the file to the place where you want to store the archive. For example, in a Linux Bash shell:

```
unzip usaspending-db.zip
```

The result of the unzip should be a directory named `usaspending-db-dump-dir`, and within that directory many files whose names are numbers, like `5284.dat.gz`, and a `toc.dat` file. This is the "directory" format of a PostgreSQL dump (e.g. `pg_dump --format=directory`).

Once unzipped, you can now inspect the contents of the archive using the `--list` flag of `pg_restore`. For example:

```
pg_restore --list usaspending-db-dump-dir
```

3. Restore the Database

This is the recommended set of shell commands to use to restore the database.

Values You May Need to Change

1. Connection variables
 - The `DBPASSWORD` variable holding the password for your `root` user. Here, it's shown as the value `password`. You can use that, but you will want to use something else if you plan on sharing access to the restored database.
 - The host and port of your Postgres server may be elsewhere, as well. So change `DBHOST` and `DBPORT` variables accordingly
 - If you are restoring to a different or existing database, you may need to change the `DBNAME` variable to something other than `data_store_api`
2. The `DUMP_DIR` variable should be the directory that you unzipped the archive into, and holds the `usaspending-db-dump-dir` directory
3. You may want to remove or comment out the refreshing of the materialized views (below the designated line in the script). Or refresh them as needed one-by-one after the base tables have been loaded.



Warnings

- **Database Dropped.** CAUTION: The initial part of the script is set up to drop and re-create a brand-new database using the `DBNAME` variable. Comment that out if you do not want to drop and re-create the database, but use an existing one. NOTE: it must connect to the `postgres` (default name) database to run drop/create database commands, so that database should exist and the `root` user should have permissions to it.
- **Materialized View Refresh Time.** If included, refreshing the materialized views takes significantly longer than restoring the base tables and their data.

- Table data load is faster because it does COPY to pull table data in bulk from the archive file. This cannot be done for materialized views, which must execute queries to get their data.

Database Restore Script

```
#### Connection Variables ####
DBUSER=root
DBPASSWORD=password
DBHOST=127.0.0.1
DBPORT=5432
CONN=postgresql://$DBUSER:$DBPASSWORD@$DBHOST:$DBPORT
DBNAME=data_store_api

#### Dump Variables ####
DUMP_DIR=/dump/data0
DUMP=$DUMP_DIR/usaspending-db-dump-dir

#### Database Restore ####

# DROP and CREATE the database, if it exists
psql $CONN/postgres -c \
    "DROP DATABASE IF EXISTS $DBNAME"
psql $CONN/postgres -c \
    "CREATE DATABASE $DBNAME"

# Create list of ALL EXCEPT materialized views data (defer them), to restore
pg_restore --list $DUMP | sed '/MATERIALIZED VIEW DATA/d' > $DUMP_DIR/restore.list

# Restore all but materialized view data
pg_restore \
    --jobs 16 \
    --dbname $CONN/$DBNAME \
    --verbose \
    --exit-on-error \
    --use-list $DUMP_DIR/restore.list \
    $DUMP

# Perform an ANALYZE to optimize query performance in view materialization
psql \
    --dbname $CONN/$DBNAME \
    --command 'ANALYZE VERBOSE;' \
    --echo-all \
    --set ON_ERROR_STOP=on \
    --set VERBOSITY=verbose \
    --set SHOW_CONTEXT=always

# =====
# ==== Comment or remove below if you do not want to materialize views ====
# =====

#### Materialized View Refresh ####

# Create list of ONLY materialized views data to refresh
pg_restore --list $DUMP | grep "MATERIALIZED VIEW DATA" > $DUMP_DIR/refresh.list

# Refresh materialized view data
pg_restore \
    --jobs 16 \
    --dbname $CONN/$DBNAME \
    --verbose \
    --exit-on-error \
    --use-list $DUMP_DIR/refresh.list \
    $DUMP

# Do an additional ANALYZE on the materialized views after being materialized
```

```
# Do an additional ANALYZE on the materialized views after being materialized
pg_restore --list $DUMP \
| grep "MATERIALIZED VIEW DATA" \
| awk '{ print "ANALYZE VERBOSE", $8";" };' \
> $DUMP_DIR/analyze_matviews.sql

psql \
--dbname $CONN/$DBNAME \
--echo-all \
--set ON_ERROR_STOP=on \
--set VERBOSITY=verbose \
--set SHOW_CONTEXT=always \
--file $DUMP_DIR/analyze_matviews.sql
```

Alternative Restore Options with `pg_restore`

The `directory` format of the dump archive is very flexible and allows you to choose what aspect of the database you want to restore. All of these options are listed in the documentation of `pg_restore` :

<https://www.postgresql.org/docs/current/app-pgrestore.html>

For example:

- If you want to restore the schema of the database and no data, you can do that with the `--schema-only` flag.
- If you want to drop all data for a table and re-add new data from this archive, you can do that with a combination of the `--table` and `--clean` flags. It will drop the table, recreate the table, and reload data for the table. (If that table has foreign key relationships you may also need `--disable-triggers` while it loads).
- If you want to pick and choose what database objects are restored -- a subset of them -- you can build manifests of what to restore using the `--list` flag. This is done in the recommended restore script above to perform the restore in stages.
 - More detail in the [Optimize Restore Order and Stages](#) section

Recommended Flags

- `--verbose` . Output verbose diagnostic messages.
- `--exit-on-error` . Stop on the first encounter of an error. Best to do this so you don't waste time on a failed load.
- `--clean` and `--if-exists` together. If you are re-loading onto an existing database, this will replace any database objects that already exists with those provided in the dump (overwrite them), if they exist.
- `--jobs 16` . This will use as many as 16 parallel workers to process the restore which can greatly speed up the time it takes.

Appendix

Example Setup of Postgres on Red Hat Enterprise Linux (RHEL)

The following steps are an example of setting up PostgreSQL on RHEL. However, the official documentation should always be consulted here:

- https://wiki.postgresql.org/wiki/YUM_Installation
- <https://www.postgresql.org/download/linux/redhat/>

```
sudo yum install -y https://download.postgresql.org/pub/repos/yum/10/redhat/rhel-7-x86_64/pgdg-re
sudo yum install -y postgresql10
sudo yum install -y postgresql10-server
```

Ensure you have the `contrib` package for PostgreSQL (see "Prerequisites" above).

```
sudo mkdir /dump/data1/pgdata
sudo chown -R postgres:postgres /dump/data1/pgdata
```

```
sudo su - postgres
export PGDATA=/dump/data1/pgdata
/usr/pgsql-10/bin/initdb --pgdata $PGDATA
/usr/pgsql-10/bin/pg_ctl -D $PGDATA start
```

Optimizing Restore Performance

Optimize Your Database Parallelism

You may be able to increase the default parallel workers for your database in the `postgresql.conf` file in the `PGDATA` directory. This will give better performance loading this dump. If you have enough cores or vCPUs, consider increasing the defaults for the following:

- `max_worker_processes` (PG10 defaults to 8)
- `max_parallel_workers_per_gather` (PG10 defaults to 2)
- `max_parallel_workers` (PG10 defaults to 8)
- If you have limited cores, you may want to lower the `--jobs` param in `pg_restore`.

Optimize Restore Order and Stages

Changing what is loaded when and in what order can help to speed up the database restore. Use the `--list` flag of `pg_restore` to create sub-sets of database objects to load, and then use the `--use-list` flag to run `pg_restore` in stages.

For instance, the step that actually materializes the data in the materialized views may move a lot faster if:

1. Table indexes are in place.
2. `ANALYZE` has been run on those tables.
3. Indexes have not yet been placed on the views.

You can experiment with alternate sequencing and staging of database objects to load using the `--list` and `--use-list` flags, and may see improvements in your restore time.